The handle http://hdl.handle.net/1887/18622 holds various files of this Leiden University dissertation.

**Author:** Vu, Van Thieu  
**Title:** Opportunities for performance optimization of applications through code generation  
**Issue Date:** 2012-03-22
Chapter 2

Automatic code generation for Finite Element Methods applied to the Shallow-Water equations

Currently, CTADEL can generate highly optimized Fortran code for several schemes in weather forecast models such as dynamics [18], cloud [16], and turbulence [42]. All these codes use finite difference methods for discretization. In this chapter we describe how to extend CTADEL to generate code for the energy conserving Galerkin finite element schemes. The advantage of finite differences is that it can be very easy to implement. However, these methods are restricted to handle rectangular shapes and simple alterations. With complex geometries and irregular physical structures, the use of finite element methods is more straightforward. Finite element methods are better suited for solving partial differential equations over complicated domains. For instance, in the simulation of the weather pattern on the earth, it is more important to have accurate predictions over land than over the wide-open sea. In order to demonstrate that finite element methods can be incorporated into the HIRLAM frame work, we have chosen to take Galerkin finite element methods to solve the Shallow-Water equations [63] as an example. These equations describe a single homogenous layer of fluid. In general they are applied to various problems, such as tidal flow, tsunamis in oceans, etc. In meteorology they are used to forecast the wind and geopotential, which is equivalent to pressure. Please note that although Galerkin finite element methods can be applied to irregular domain, in the context of its use in meteorology only regular domains are chosen in this chapter.
2.1 Galerkin finite element methods

This section gives a short description of Galerkin finite element methods for the Shallow-Water equations. For more details, the reader is referred to [72, 73, 75].

2.1.1 Galerkin projection and finite element methods

For an arbitrary field $\phi$, we assume the following representation

$$\tilde{\phi}(r) = \sum_{\nu=1}^{N} \phi_{\nu} e_{\nu}(r).$$  \hspace{1cm} (2.1)

Here $r$ denotes the $x$ or $y$ horizontal coordinates, $e_{\nu}$ are basis functions, $N$ is a degree and $\phi_{\nu}$ are coefficients of approximation.

In order to formulate the finite element methods, it is necessary to approximate a general field $\phi(r)$ by a function $\tilde{\phi}(r)$ in the form of the right hand side of Equation (2.1). The transformation from $\phi(r)$ to $\tilde{\phi}(r)$ can be achieved by the Galerkin projection as

$$\tilde{\phi} = (\phi_{\nu}, e_{\nu}), \; \nu \in \{1, \cdots, N\},$$  \hspace{1cm} (2.2)

where the scalar product $(a, b)$ is computed as

$$(a, b) = \int a(r) b(r) d\mu,$$  \hspace{1cm} (2.3)

with

$$d\mu = w(r) \, dx \, dy.$$  \hspace{1cm} (2.4)

The positive continuous function $w$ is called the weight of the Galerkin projection. The relation between the fields $\phi$ and $\tilde{\phi}$, given by (2.2), can be written as

$$\tilde{\phi} = G(\phi),$$  \hspace{1cm} (2.5)

where $G$ is called the Galerkin projection operator. The angle brackets denote the Galerkin projection.

For equations of the form

$$\phi_t = f(\phi),$$  \hspace{1cm} (2.6)

where the subscript $t$ denotes the differentiation in time, the approximation

$$\tilde{\phi}_t = G(f(\tilde{\phi}))$$  \hspace{1cm} (2.7)

is called the standard finite element scheme.
2.1.2 The Galerkin finite element method for the Shallow-Water equations

The Shallow-Water equations model the propagation of disturbances in water and other incompressible fluids. The underlying assumption is that the depth of the fluid is small compared to the wave length of the disturbance. The equations are derived from the principles of conservation of mass and conservation of momentum. The independent variables are time \( t \) and two space coordinates \( x \) and \( y \). The dependent variables are the fluid height or depth \( H \), and the two-dimensional fluid velocity field with components \( U \) and \( V \) in the \( x \) and \( y \) direction, respectively. With the proper choice of units, the conserved quantities are mass, which is proportional to \( H \), and momentum, which is proportional to \( UH \) and \( VH \). The force acting on the fluid is gravity, represented by the gravitational constant \( g \). The Shallow-Water equations have the following form

\[
\begin{align*}
\frac{\partial U}{\partial t} &= fV - UU_x - VU_y - gH_x, \\
\frac{\partial V}{\partial t} &= -fU - UV_x - VV_y - gH_y, \\
\frac{\partial H}{\partial t} &= -\left( UH \right)_x - \left( VH \right)_y,
\end{align*}
\]  

(2.8)

where the subscripts denote the differentiation with respect to the specified variable, and \( f \) represents the Coriolis parameter, describing the rotation of the earth.

The energy conserving scheme for the Shallow-Water equations, as proposed by Steppeler [73, 75, 74], reads as follows

\[
\begin{align*}
\frac{\partial U}{\partial t} &= G_1 \left\langle \eta V - \left( G_2 \left\langle \frac{1}{2} \left( U^2 + V^2 \right) + gH \right\rangle \right)_x \right\rangle, \\
\frac{\partial V}{\partial t} &= G_1 \left\langle -\eta U - \left( G_2 \left\langle \frac{1}{2} \left( U^2 + V^2 \right) + gH \right\rangle \right)_y \right\rangle, \\
\frac{\partial H}{\partial t} &= -G_2 \left\langle \left( UH \right)_x + \left( VH \right)_y \right\rangle, \\
\eta &= V_x - U_y + f,
\end{align*}
\]

(2.9)

where \( \eta \) represents the absolute vorticity. \( G_1 \) and \( G_2 \) are the Galerkin operators with weights \( H \) and 1, respectively.

To approximate Equation (2.9) by the Galerkin finite element method, we interpolate the fields from the nodepoint to the \( y \)-collocation grid and then from the \( y \)-collocation to the collocation grid (see Figure 2.1). The purpose of interpolation is to create a finer resolution grid. Next, we compute the right hand side of Equation (2.9) on the collocation grid. Finally, we perform the
Figure 2.1: Nodepoint grid (left); y-collocation grid (middle), obtained by interpolating the nodepoint grid in the y direction; and collocation grid (right), obtained by interpolating the y-collocation grid in the x direction. $\nu$ and $\mu$ are the coordinates of the nodepoint grid. Right arrow shows the interpolation. Left arrow shows the projection.

Galerkin projection in the $x$ direction and then in the $y$ direction to transform from the collocation to the nodepoint grid. In the next paragraphs, we will describe these interpolations and projections.

**Interpolation**

Let $\xi$ be either $x$ or $y$ and $\psi(\xi)$ be a function obtained from a two dimensional function $\psi(x,y)$ by keeping the other variable fixed. Then, an interpolation formula for an interval $(a,b)$ can be written as

$$\psi(\xi) = \frac{\psi(a) (b - \xi) + \psi(b) (\xi - a)}{b - a}, \quad (2.10)$$

where $\psi$ stands for $U$, $V$, or $H$. Within each interval, we always interpolate to the three Gauss-points $\xi_{\rho}$, $\rho = 1, 2, 3$ [1], defined as

$$\xi_{\rho} = \frac{b - a}{2} \xi'_{\rho} + \frac{b + a}{2}, \quad (2.11)$$

with

$$\xi'_{1,3} = \mp \sqrt{\frac{3}{5}}, \quad \xi'_{2} = 0. \quad (2.12)$$

Choosing the interval $(a,b) \equiv (\xi, \xi + 1)$ and substitute Equation (2.11) in Equation (2.10), we have

$$\psi(\xi_{\rho}) = \frac{\psi(\xi) \left(1 - \xi_{\rho}\right) + \psi(\xi + 1) \left(1 + \xi'_{\rho}\right)}{2}. \quad (2.13)$$
2.1. Galerkin finite element methods

Let
\[
\psi(\xi)^+ = \frac{\psi(\xi) + \psi(\xi + 1)}{2},
\]
\[
\Delta^+ \psi(\xi) = \psi(\xi + 1) - \psi(\xi)
\]
(2.14)
then Equation (2.13) can be rewritten as
\[
\psi(\xi_\rho) = \psi(\xi)^+ + \frac{1}{2} \xi_\rho \Delta^+ \psi(\xi).
\]
(2.15)

This equation is used for interpolation. With \(\xi = y\) and taking \((\nu, \nu + 1)\) as interval, where \(\nu\) is the coordinate in the \(x\)-direction of the nodepoint grid, the interpolation will transform from the nodepoint to the \(y\)-collocation grid. Similarly, interpolation with \(\xi = x\) in the interval \((\mu, \mu + 1)\), where \(\mu\) is the coordinate in the \(y\)-direction of the nodepoint grid, will transform fields from the \(y\)-collocation to the collocation grid (see Figure 2.1).

Projection

Let \(RS(\xi)\) denote the right hand side of Equation (2.9) where \(\xi\) is either \(x\) or \(y\). Then the Galerkin projection for \(RS(\xi)\) is defined as
\[
G \langle RS(\xi) \rangle = \int e_\nu(\xi_\mu) RS(\xi) w(\xi) d\xi,
\]
(2.16)
with basis functions \(e_\nu(\xi_\mu)\) defined by the properties
\[
e_\nu(\xi_\mu) = \begin{cases} 
0, & \text{if } \mu \neq \nu, \\
1, & \text{if } \mu = \nu,
\end{cases}
\]
(2.17)
and \(w(\xi)\) is a weight function which is either \(H\) or 1 for the \(G_1\) and \(G_2\) Galerkin operator, respectively.

The integration of any function \(\psi(\xi)\) in the interval \((a, b)\) can be computed as follows
\[
\int_a^b \psi(\xi) d\xi = \sum_{\rho=1}^3 \psi(\xi_\rho) g_\rho (b - a),
\]
(2.18)
with \(\xi_\rho\) are calculated according to Equation (2.11) and \(g_\rho\) are the Gaussian-weights, defined as
\[
g_{1,3} = 0.55555555555, \\
g_2 = 0.88888888889.
\]
(2.19)
Choosing the interval \((a, b) \equiv (\xi, \xi + 1)\) and substitute Equations (2.17) and (2.18) in Equation (2.16), we obtain

\[
\psi(\xi) = \frac{\Delta \xi}{4} \sum_{\rho=1}^{3} \left( \psi(\xi_{\rho}) w(\xi_{\rho}) g_{\rho} \left( 1 - \xi_{\rho}' \right) + \psi(\xi_{\rho} - 1) w(\xi_{\rho} - 1) g_{\rho} + \xi_{\rho}' \psi(\xi_{\rho} - 1) w(\xi_{\rho} - 1) g_{\rho} \right).
\]

(2.20)

Let

\[
\psi(\xi_{\rho}) w(\xi_{\rho}) = \frac{\psi(\xi_{\rho} - 1) w(\xi_{\rho} - 1) + \psi(\xi_{\rho}) w(\xi_{\rho})}{2}
\]

(2.21)

and

\[
\Delta^{-} \psi(\xi_{\rho}) w(\xi_{\rho}) = \psi(\xi_{\rho}) w(\xi_{\rho}) - \psi(\xi_{\rho} - 1) w(\xi_{\rho} - 1),
\]

(2.22)

we get

\[
\psi(\xi) = \frac{\Delta \xi}{2} \sum_{\rho=1}^{3} g_{\rho} \left( \psi(\xi_{\rho}) w(\xi_{\rho}) - \frac{\xi_{\rho}'}{2} \Delta^{-} \psi(\xi_{\rho}) w(\xi_{\rho}) \right).
\]

(2.23)

Equation (2.23) is used for projection. If \(\xi = x\) and \(\psi\) given on the collocation grid, by keeping \(y\) fixed, the Galerkin projection will transform from the collocation to the \(y\)-collocation grid. Similarly, if \(\xi = y\) and \(\psi\) is given on the \(y\)-collocation grid, by keeping \(x\) fixed, the Galerkin projection will transform from the \(y\)-collocation to the nodepoint grid.

### 2.2 Specification

CTADEL generates code from input specifications. The problem, specified in a high-level language, is transformed into an Abstract Syntax Tree. Then common subexpression elimination and architecture dependent optimizations are performed before the resulting code is generated. In this section we present how to extend CTADEL for the Galerkin finite element methods. The Galerkin finite element methods consist of two main steps: interpolation and projection. To simplify the implementation, we specify each step as a template. Next, we show the specification of the Shallow-Water equations applying the Galerkin finite element method.

#### 2.2.1 Templates

**Interpolation template**

The interpolation of a field is performed following Equation (2.15). In CTADEL we define the template \textit{interpolate} which has as input \(\psi\), the field to be interpolated in the \(\xi\) direction, and as output the corresponding interpolated field. The code of \textit{interpolate} reads
2.2. Specification

average_f(_:::field Xi(grid),Xi) :: field Xi(half)
average_f(Psi::float dim Unit,Xi) :: float dim Unit
:= 1/2*(Psi+Psi0(Xi=Xi+1)).

delta_f(_:::field Xi(grid),Xi) :: field Xi(half).
delta_f(Psi::float dim Unit,Xi) :: float dim Unit := Psi0(Xi=Xi+1)-Psi.

interpolate(_:::field Xi(grid),_:::field Xi(grid),Xi) :: field Xi(grid).
interpolate(Psi::float dim Unit,Gp::float dim Unit,Xi)::float dim Unit := average_f(Psi,Xi)+1/2*Gp*delta_f(Psi,Xi).

In the template interpolate we denote average_f, delta_f, Psi, Gp, and Xi for the notations $\psi^+$, $\triangle^+$, $\psi$, $\xi'$, and $\xi$ as in Equation (2.15), respectively. The \TeX output for this template produced by CTADEL has the following form

\[
\text{interpolate} \left( \psi, \xi', \rho \right) = \psi(\xi') + \frac{1}{2} \xi' \triangle^+ \psi(\xi). \tag{2.24}
\]

We see that the right hand sides of Equations (2.15) and (2.24) are exactly the same. This proves that the template \textit{interpolate} definition is correct.

Projection template

Based on Equation (2.23) we define the template \textit{project} for Galerkin projection. This template has as input the field $\psi(\xi_\rho)$ to be projected in the $\xi$ direction, the weight function $w(\xi_\rho)$, the Gauss-points $\xi'_\rho$ and the Gauss-weights $g_\rho$ parameters. The output of \textit{project} is the corresponding projected field. The template \textit{project} has the following form

average_b(_:::field Xi(grid),Xi) :: field Xi(half).
average_b(Psi::float dim Unit,Xi):: float dim Unit
:= 1/2*(Psi0(Xi=Xi-1)+Psi).

delta_b(_:::field Xi(grid),Xi) :: field Xi(half).
delta_b(Psi::float dim Unit,Xi) :: float dim Unit := Psi-Psi0(Xi=Xi-1).

project(_:::field Xi(grid),_:::field Xi(grid),_:::field Xi(grid),
_:::field Xi(grid),Xi) :: field Xi(grid).
project(Psi::float dim Unit,W::float dim Unit,Gp::float dim Unit,
Gw::float dim Unit,Xi)::float dim Unit
:= sum(1/2*delta(Xi)*Gw*(average_b((PsiW),Xi)
-1/2*Gp*delta_f((PsiW),Xi)),rho = 1..3).

In the template \textit{project} average_b and delta_b denote $\overline{\psi}^-$ and $\overline{\triangle}^-$, respectively. Psi is the input field $\psi(\xi_\rho)$, Xi is direction of projection $\xi$, W is the weight function $w$, and Gp and Gw denote the Gauss-points $\xi'_\rho$ and the Gauss-weights
Chapter 2. Automatic code generation for Finite Element Methods

g_{p}, respectively. CTADEL produces the \LaTeX output for this template as

\[
\text{project}\left(\psi, w, \xi'_{p}, g_{p}, \xi\right) = \frac{\Delta \xi}{2} \sum_{p=1}^{3} g_{p} \left( \frac{\psi(\xi_{p}) w(\xi_{p})}{\psi(\xi_{p}) w(\xi_{p})} - \frac{\xi'_{p}}{2} \psi(\xi_{p}) w(\xi_{p}) \right).
\] \hspace{1cm} (2.25)

The right hand side of Equation (2.25) is exactly the same as in Equation (2.23). This proves the correctness of the project template definition.

2.2.2 Specification of the Shallow-Water equations

We give the full specification of the Shallow-Water equations. The specification consists of the definitions of units and dimensions, the definitions of grids, and the discretization of the Shallow-Water equations. Note that in this chapter we have chosen the rectangular grids. Based on this specification, CTADEL will generate Fortran code.

Units and dimensions

At first we need to define all units and dimensions that are used for describing the Shallow-Water equations such as Distance (m), Time (s), Velocity (Distance/Time), and Height (Distance). These units and dimensions are specified in CTADEL as

```plaintext
% Define units and dimensions
Distance : "m".
Time : "s".
[Velocity] := [Distance]/[Time].
[Height] := [Distance].
```

% Spatial variables
x :: float dim Distance. % for nodepoint grid
y :: float dim Distance.
xc :: float dim Distance. % for collocation grid
yc :: float dim Distance.

In the above specification, \((x,y)\) and \((xc,yc)\) are the spatial variables used to discretize fields on the nodepoint and the collocation grid, respectively.

Grids

Before defining the grids, we need to declare the grid sizes. The grid sizes, which are defined by \(L\) and \(LCOL\) for the nodepoint and collocation grid, respectively, are specified as

2.2. Specification

% Grid sizes
L :: natural. % for nodepoint grid
LCOL :: natural. % for collocation grid

The grids that are used to discretize the Shallow-Water equations include the
nodepoint, y-collocation, and collocation grid. The nodepoint grid has two
dimension variables denoted as longitudinal and latitudinal. The y-collocation
grid is extended from the nodepoint grid by the sub-dimension denoted as
latit_collocation. The collocation grid is extended from the y-collocation grid
with the sub-dimension denoted as longi_collocation. In CTADEL the nodepoint,
y-collocation, and collocation grids are specified as

% Define nodepoint grid
longitudinal :=i=1 .. L.
latitudinal :=j=1 .. L.
nodepoint :=longitudinal by latitudinal.

% Define y-collocation grid
latit_collocation :=jc=1 .. LCOL.
y_collocation :=nodepoint by latit_collocation.

% Define collocation grid
longi_collocation :=ic=1 .. LCOL.
collocation :=y_collocation by longi_collocation.

The notation of a domain consists of a variable and an associated range of
grid sizes. Specifically, nodepoint denotes grid point (i,j) on a [1,L]×[1,L]
domain, y-collocation denotes grid point (i,j,jc) on a [1,L]×[1,L]×[1,LCOL] do-
main, and collocation denotes grid point (i,ic,jc,jc) on a [1,L]×[1,LCOL]×[1,L]×
[1,LCOL] domain.

Discretization
Before discretizing the Shallow-Water equations by the Galerkin finite element
method, we need to declare all parameters such as Coriolis F, Gauss-points Gp,
and Gauss-weights Gw. These parameters are specified as

% Coriolis parameter
F :: float dim "1/s".

% Gauss-points (Gp), Gauss-weights (Gw)
Gp::float field (xc(grid)) on longi_collocation.
Gw::float field (xc(grid)) on longi_collocation.

Next we specify all inputs (U,V,H), outputs (UT,VT,HT) of the Shallow-
Water equations, and intermediate values on the nodepoint, y-collocation, and
collocation grids. In the following specification, we denote the nodepoint, y-collocation, and collocation grid by node, ycol, and col, respectively. For examples Unode, Uycol, and Ucol represent the field $U$ on the nodepoint, y-collocation, and collocation grid, respectively.

\% Inputs $U$, $V$, and $H$ are specified:
\% on nodepoint grid:
  Unode::float dim Velocity field (x(grid),y(grid),t) on nodepoint.
  Vnode::float dim Velocity field (x(grid),y(grid),t) on nodepoint.
  Hnode::float dim Height field (x(grid),y(grid),t) on nodepoint.
\% on y-collocation grid:
  Uycol::float dim Velocity field (x(grid),y(grid),yc(grid),t) on y_collocation.
 Vycol::float dim Velocity field (x(grid),y(grid),yc(grid),t) on y_collocation.
  Hycol::float dim Height field (x(grid),y(grid),yc(grid),t) on y_collocation.
\% on collocation grid:
  Ucol ::float dim Velocity field (x(grid),xc(grid),y(grid),yc(grid),t) on collocation.
  Vcol ::float dim Velocity field (x(grid),xc(grid),y(grid),yc(grid),t) on collocation.
  Hcol ::float dim Height field (x(grid),xc(grid),y(grid),yc(grid),t) on collocation.
\% Outputs UT, VT, and HT are specified:
\% on nodepoint grid:
  UTnode::float dim Velocity field (x(grid),y(grid),t) on nodepoint.
  VTnode::float dim Velocity field (x(grid),y(grid),t) on nodepoint.
  HTnode::float dim Height field (x(grid),y(grid),t) on nodepoint.
\% on y-collocation grid:
  UTycol::float dim Velocity field (x(grid),y(grid),yc(grid),t) on y_collocation.
  VTycol::float dim Velocity field (x(grid),y(grid),yc(grid),t) on y_collocation.
  HTycol::float dim Height field (x(grid),y(grid),yc(grid),t) on y_collocation.
\% on collocation grid:
  UTcol ::float dim Velocity field (x(grid),xc(grid),y(grid),yc(grid),t) on collocation.
2.2. Specification

\[ VT_{\text{col}} :: \text{float dim Velocity field } (x(\text{grid}), x_c(\text{grid}), y(\text{grid}), y_c(\text{grid}), t) \]
on collocation.

\[ HT_{\text{col}} :: \text{float dim Height field } (x(\text{grid}), x_c(\text{grid}), y(\text{grid}), y_c(\text{grid}), t) \]
on collocation.

% Intermediate values \( UVH=G_2 <\frac{1}{2}*(U^2+V^2)+gH> \) and \( XI \) are specified:
% on nodepoint grid:
\[ UVH_{\text{node}} :: \text{float dim Velocity field } (x(\text{grid}), y(\text{grid}), t) \]
on nodepoint.
% on y-collocation grid:
\[ UVH_{\text{ycol}} :: \text{float dim Velocity field } (x(\text{grid}), y(\text{grid}), y_c(\text{grid}), t) \]
on y_collocation.
% on collocation grid:
\[ UVH_{\text{col}} :: \text{float dim Velocity field } (x(\text{grid}), x_c(\text{grid}), y(\text{grid}), y_c(\text{grid}), t) \]
on collocation.
\[ XI_{\text{col}} :: \text{float dim Velocity field } (x(\text{grid}), x_c(\text{grid}), y(\text{grid}), y_c(\text{grid}), t) \]
on collocation.

We note that we solve the Shallow-Water equations in a number of time steps. Therefore, the above specification involves the time variable \( t \).

The \textit{interpolate} template defined in Subsection 2.2.1 is used to transform inputs from the nodepoint grid to the y-collocation and collocation grid. The interpolation of inputs are specified as

% Interpolate inputs from nodepoint to y-collocation.
\[ U_y_{\text{col}} = \text{interpolate}(U, Gp, y). \]
\[ V_y_{\text{col}} = \text{interpolate}(V, Gp, y). \]
\[ H_y_{\text{col}} = \text{interpolate}(H, Gp, y). \]
% Interpolate inputs from y-collocation to collocation grid.
\[ U_{\text{col}} = \text{interpolate}(U_{y_{\text{col}}}, Gp, x). \]
\[ V_{\text{col}} = \text{interpolate}(V_{y_{\text{col}}}, Gp, x). \]
\[ H_{\text{col}} = \text{interpolate}(H_{y_{\text{col}}}, Gp, x). \]

We continue with the specification of the expression \( G2<\frac{1}{2}*(U^2+V^2)+gH> \), which is used twice in the calculation of \( UT \) and \( VT \). For convenience, we denote this expression by \( UVH \). The specification of \( UVH \) reads as

% Compute expression: \( UVH = - G2<\frac{1}{2}*(U^2+V^2)+gH> \)
% Compute UVH on collocation grid
\[ UVH_{\text{col}} = -\frac{1}{2}*(U_{\text{col}}^2+V_{\text{col}}^2) - gH_{\text{col}}. \]

% Project UVH from collocation to the nodepoint grid
\[ UVH_{\text{ycol}} = \text{project\_bound}(UVH_{\text{col}}, 1, Gp, Gw, x) \text{ if on x-boundary}\]
\[ \text{project}(UVH_{\text{col}}, 1, Gp, Gw, x) \text{ otherwise}. \]
\[ UVH = \text{project\_bound}(UVH_{\text{ycol}}, 1, Gp, Gw, y) \text{ if on y-boundary}\]
\[ \text{project}(UVH_{\text{ycol}}, 1, Gp, Gw, y) \text{ otherwise}. \]
% Compute the derivative of UVH in the y-direction
% (UVHcol_y = d(UVH)/dy) on the y_collocation grid
% and interpolate it to collocation grid
UVHcol_y::float dim Velocity field (x(grid),y(grid),yc(grid),t)
on y_collocation.

UVHcol_x::float dim Velocity field(x(grid),xc(grid),y(grid),yc(grid),t)
on collocation.

UVHcol_y = d(UVH)/dy.
UVHcol_y = interpolate(UVHcol_y,Gp,x).

% Compute the derivative of UVH in the x-direction
% (UVHcol_x = d(UVH)/dx) on the collocation grid

UVHcol_x::float dim Velocity field(x(grid),xc(grid),y(grid),yc(grid),t)
on collocation.

UVHcol_x = d(UVHcol)/dx.

In the above specification, dX/dY stands for the derivative of variable X to variable Y. Its only use is where Y is a spatial variable. It is approximated by a finite difference:

$$\frac{dX}{dY} = \frac{X_{i+1} - X_i}{\Delta Y},$$

(2.26)

where \(\Delta Y\) is the grid point distance in Y direction, \(X_i\) and \(X_{i+1}\) denote values of \(X\) at grid point \(i\) and \(i+1\), respectively. Within CTADEL, Equation (2.26) is specified as:

d (_ :: field Y(grid)) / d (_ :: field Y(grid)) :: field Y(grid).

d (X :: float dim Unit) / d (Y :: float dim Unit) :: float dim Unit
:= (X @ (Y = Y+1) - X) / delta Y.

Finally, we specify the Shallow-Water equations on the collocation grid and the projection of the outputs \(UT\), \(VT\), and \(HT\) from the collocation grid to the y-collocation and nodepoint grid as

% Compute outputs of the Shallow-Water equations on collocation grid

UTcol = XI*Vcol + UVHcol_x.
VTcol = -XI*Ucol + UVHcol_y.
HTcol = -d(Ucol*Hcol)/dx - d(Vcol*Hcol)/dy.
XI = d(Vcol)/dx - d(Ucol)/dy + F.

% Project outputs from collocation to y-collocation and nodepoint grid

UTycol = project_bound(UTcol,H,Gp,Gw,x) if on x-boundary\n       project(UTcol,H,Gp,Gw,x) otherwise.
UT = project_bound(UTycol,H,Gp,Gw,y) if on y-boundary\n      project(UTycol,H,Gp,Gw,y) otherwise.
2.3 Generating code

This section presents the code generation process in CTADEL. We describe the steps that CTADEL performs in order to translate the abstract specification to optimized Fortran code. We give the \LaTeX output of each step for the field \(H\) which is the result of the specification given in Subsection 2.2.2. These \LaTeX outputs are produced by CTADEL to compare the specification with the mathematical description and to check the different steps. The outputs of the fields \(U\) and \(V\) are similar.

In CTADEL, the code is generated through the following steps:

1. Get scalar values and dimensional analysis: All scalar values and equations in the specification are retrieved and sorted in order of computation. Then CTADEL analyzes all units and dimensions declared in the specification. The result of this step is given in Figure 2.2. In this figure, \(U_{\text{col}}, V_{\text{col}}, \text{and } H_{\text{col}}\) represent the interpolation values of \(U\), \(V\), and \(H\) on the collocation grid, respectively. \(H_{T\text{col}}, H_{T_{\text{ycol}}, \text{and } H_{T\text{col}}\text{ denote the interpolation values of the derivative in time of } H/\partial t\text{ on the collocation, y-collocation, and nodepoint grid, respectively. } \Delta X\text{ and } \Delta Y\text{ are the grid point distances on the nodepoint grid in the } x\text{ and } y\text{ direction, respectively. In Figure 2.2, the first equation shows the calculation of } H_{T\text{ on the collocation grid. Comparing to Equation 2.9, we confirm the correctness of the Shallow-Water equations specification. The second and third equations show the projection of } H_{T\text{ from the collocation grid to the y-collocation and nodepoint grid, respectively. The projection at the boundary requires extra information. Hence, the boundary condition is included in the second and third equations.}

2. Discretization: Discretization of the continuous equations by the finite element method. Interpolation and Galerkin projection are used to discretize all spatial variables and their derivatives. The output of this step consists of the spatial discrete equations as given in Figure 2.3. Since extra information is needed for discretizing at the boundary, the boundary
Chapter 2. Automatic code generation for Finite Element Methods

\[ HT_{col} := -\frac{\partial (U_{col} H_{col})}{\partial x} - \frac{\partial (V_{col} H_{col})}{\partial y} \]

\[ HT_{ycol} := \begin{cases} \sum_{\rho=1}^{3} \left( \frac{\Delta X}{2} g_{\rho} \left( (HT_{col})_{\text{bound}} - \frac{\xi_{\rho}'}{2} \Delta_{\text{down}} HT_{col} \right) \right) & \text{if on}_x \text{ bound} \\ \sum_{\rho=1}^{3} \left( \frac{\Delta X}{2} g_{\rho} \left( (HT_{col}) - \frac{\xi_{\rho}'}{2} \Delta_{\text{down}} HT_{col} \right) \right) & \text{if on}_y \text{ bound} \end{cases} \]

\[ HT := \begin{cases} \sum_{\rho=1}^{3} \left( \frac{\Delta Y}{2} g_{\rho} \left( (HT_{ycol})_{\text{bound}} - \frac{\xi_{\rho}'}{2} \Delta_{\text{down}} HT_{ycol} \right) \right) & \text{if on}_x \text{ bound} \\ \sum_{\rho=1}^{3} \left( \frac{\Delta Y}{2} g_{\rho} \left( (HT_{ycol}) - \frac{\xi_{\rho}'}{2} \Delta_{\text{down}} HT_{ycol} \right) \right) & \text{if on}_y \text{ bound} \end{cases} \]

\[ HT_{col} \]

\[ HT_{ycol} := \begin{cases} \sum_{\rho=1}^{3} \left( \frac{\Delta X}{2} g_{\rho} \left( (HT_{col})_{\text{bound}} - \frac{\xi_{\rho}'}{2} \Delta_{\text{down}} HT_{col} \right) \right) & \text{if on}_x \text{ bound} \\ \sum_{\rho=1}^{3} \left( \frac{\Delta X}{2} g_{\rho} \left( (HT_{col}) - \frac{\xi_{\rho}'}{2} \Delta_{\text{down}} HT_{col} \right) \right) & \text{if on}_y \text{ bound} \end{cases} \]

\[ HT := \begin{cases} \sum_{\rho=1}^{3} \left( \frac{\Delta Y}{2} g_{\rho} \left( (HT_{ycol})_{\text{bound}} - \frac{\xi_{\rho}'}{2} \Delta_{\text{down}} HT_{ycol} \right) \right) & \text{if on}_x \text{ bound} \\ \sum_{\rho=1}^{3} \left( \frac{\Delta Y}{2} g_{\rho} \left( (HT_{ycol}) - \frac{\xi_{\rho}'}{2} \Delta_{\text{down}} HT_{ycol} \right) \right) & \text{if on}_y \text{ bound} \end{cases} \]

\[ \text{Figure 2.2:} \text{\LaTeX\ output generated by CTADEL after get scalar values and dimensional analysis} \]

condition is applied in this process. For example, the calculation of \( HT_{i,j} \) needs the information of \( HT_{col_{i-1,ic,jc}} \) (shaded in Figure 2.3). Hence, to calculate \( HT \) at the left boundary in the \( x \)-direction \((i = 1)\), we need the extra information of \( HT_{col} \). Because we apply the periodic boundary condition in the \( x \)-direction for the Shallow-Water equations, the value \( HT_{col_{i-1,ic,jc}} \) at \( i = 1 \) is replaced by \( HT_{col_{L-1,ic,jc}} \). This example has been marked by shading.

3. Common-subexpression elimination (CSE): The global CSE constructs a dataflow graph for the intermediate code which is iteratively refined using a hardware cost model of the target computer architecture. The hardware cost model describes the relative cost of arithmetic and memory operations of the hardware resource. The result of CSE is an optimized intermediate code in which the subexpressions which are repeatedly used are computed and stored in temporary variables for later usages. After CSE, CTADEL gives the output as in Figure 2.4. In this figure, temporary variables are denoted as “\( T \)” followed by a number. To compute \( HT \), 11 temporary variables have been created. The use of temporary variables increases the efficiency of computation. However, it results in the use of more memory.

4. Code generation: From the optimized intermediate code, CTADEL produces the Fortran code based on predefined templates. The code generated by CTADEL is given in Figure 2.5. For brevity, we show only the code that relates to the calculations of \( HT_{col} \) and \( HT \) as in Figure 2.4.
2.3. Generating code

\[ R_{icol} := \begin{cases} 
7.954545454545454545E - 7 * \\
\frac{1}{2} \left( H_{i+1,j} \ast (1 - Gp_{jc}) + H_{i+1,j+1} \ast (Gp_{jc} + 1) \right) \\
- \frac{1}{2} \left( H_{i,j} \ast (1 - Gp_{jc}) + H_{i,j+1} \ast (Gp_{jc} + 1) \right) \\
+ \frac{1}{2} \ast (Gp_{jc} + 1) \ast (U_{i+1,j} \ast (1 - Gp_{jc}) + U_{i+1,j+1} \ast (Gp_{jc} + 1)) \\
\end{cases} \]

\[ = \begin{cases} 
1/4 \ast \left( \frac{1}{2} \ast (1 - Gp_{jc}) \ast (U_{i,j} \ast (1 - Gp_{jc}) + U_{i,j+1} \ast (Gp_{jc} + 1)) \right) \\
+ 1/4 \ast (Gp_{jc} + 1) \ast (U_{i,j} \ast (1 - Gp_{jc}) + U_{i,j+1} \ast (Gp_{jc} + 1)) \\
\end{cases} \]

\[ \begin{cases} 
1.590909090909090909E - 6 \ast (1 - Gp_{jc}) \ast (H_{i,j+1} - H_{i,j}) + \\
1.590909090909090909E - 6 \ast (Gp_{jc} + 1) \ast (H_{i,j+1} - H_{i,j}) \\
+ 1/2 \ast (Gp_{jc} + 1) \ast (V_{i+1,j} \ast (1 - Gp_{jc}) + V_{i+1,j+1} \ast (Gp_{jc} + 1)) \\
\end{cases} \]

\[ \begin{cases} 
1.590909090909090909E - 6 \ast \\
1/2 \ast (1 - Gp_{jc}) \ast (H_{i,j} \ast (1 - Gp_{jc}) + H_{i,j+1} \ast (Gp_{jc} + 1)) \\
+ 1/2 \ast (Gp_{jc} + 1) \ast (H_{i,j} \ast (1 - Gp_{jc}) + H_{i,j+1} \ast (Gp_{jc} + 1)) \\
\end{cases} \]

\[ \begin{cases} 
1.5714285714E5 \ast (Gw_{ic} \ast (1 - Gp_{jc})) \\
+ \text{HTcol} \quad (\text{if } i = 1) \\
1.5714285714E5 \ast (H_{icol} \ast (1 - Gp_{jc})) \\
\text{HTcol} \quad (\text{if } j = 1) \\
\end{cases} \]

\[ \begin{cases} 
1.5714285714E5 \ast (H_{icol} \ast (1 - Gp_{jc})) \\
+ \text{HTcol} \quad (\text{if } i = 1) \\
1.5714285714E5 \ast (H_{icol} \ast (1 - Gp_{jc})) \\
\text{HTcol} \quad (\text{if } j = 1) \\
\end{cases} \]

\[ \begin{cases} 
1.5714285714E5 \ast (H_{icol} \ast (1 - Gp_{jc})) \\
1.5714285714E5 \ast (H_{icol} \ast (1 - Gp_{jc})) \\
+ \text{HTcol} \quad (\text{if } i = 1) \\
+ \text{HTcol} \quad (\text{if } j = 1) \\
\end{cases} \]

\[ \text{Figure 2.3: } \text{\LaTeX} \text{ output generated by CTADEL after discretization. The shading shows an example of applying the periodic boundary condition.} \]
\( t_{0,jc} := 1 - G_{p,jc} \)
\( t_{2,jc} := G_{p,jc} + 1 \)
\( t_{7,ie} := 1 - G_{p,ie} \)
\( t_{12,ie} := G_{p,ie} + 1 \)
\( t_{13,i,j,je} := 1/2 \ast (U_{i,j} \ast t_{0,je} + U_{i,j+1} \ast t_{2,je}) \)
\( t_{19,i,j,je} := 1.5909090909090909E - 6 \ast (H_{i,j+1} - H_{i,j}) \)
\( t_{26,i,j,je} := 1/2 \ast (V_{i,j} \ast t_{0,je} + V_{i,j+1} \ast t_{2,je}) \)
\( t_{34,i,j,je} := 1.5909090909090909E - 6 \ast (V_{i,j+1} - V_{i,j}) \)
\( t_{5,i,j,je} := 1/2 \ast (H_{i,j} \ast t_{0,je} + H_{i,j+1} \ast t_{2,je}) \)

\[
HT_{col,i,ie,j,je} := - \left( 7.9545454545454549E - 7 \ast (t_{5,i+1,j,jc} - t_{5,i,j,je}) \right) \\
- \left( \frac{1}{4} \ast \left( t_{12,ie} \ast t_{13,i+1,j,je} + t_{13,i,j,je} \ast t_{7,ie} \right) \right) \\
- \left( \frac{1}{2} \ast \left( 1.5909090909090909E - 6 \ast \left( t_{13,i+1,j,je} - t_{13,i,j,je} \right) \right) \right) \\
+ \frac{1}{2} \ast \left( t_{12,ie} \ast t_{34,i+1,j,je} + t_{34,i,j,je} \ast t_{7,ie} \right) \\
+ \frac{1}{2} \ast \left( t_{12,ie} \ast t_{5,i+1,j,je} + t_{5,i,j,je} \ast t_{7,ie} \right)
\]

\( t_{46,i,ic,j,je} := HT_{col,i,ic,j,je} \ast G_{w,ie} \)
\( t_{102,i,ic,j,je} := G_{w,je} \ast 1.5714285714E5 \)

\[
\begin{align*}
(t_{12,ie} \ast t_{46,i-1,ic,j,je} + t_{46,i,ic,j,je} \ast t_{7,ie}) & \quad \text{if } i = 1 \\
(t_{12,ie} \ast t_{46,i-1,ic,j,je} + t_{46,i,ic,j,je} \ast t_{7,ie}) & \quad \text{if } i \neq 1
\end{align*}
\]

\[
HT_{i,j} := \begin{cases} 
1.5714285714E5 \ast (t_{0,je} \ast t_{102,i,ic,j,je}) & \text{if } j = 1 \\
1.5714285714E5 \ast (t_{0,je} \ast t_{102,i,ic,j,je} + t_{2,je} \ast t_{102,i,ic,j-1,je}) & \text{if } j \neq 1
\end{cases}
\]

**Figure 2.4:** \LaTeX\ output generated by CTADEL after CSE
2.3. Generating code

DO 119 jc=1,LCOL
  DO 120 j=0,L
    DO 121 ic=1,LCOL
      DO 122 i=0,L
        HTcol(i,ic,j,jc)=(-7.95454545454549E-7*
          (t5(i+1,j,jc)-t5(i,j,jc))*(t12(ic)*
          t13(i+1,j,jc)+t13(i,j,jc)*t7(ic))-2.5E-1*
          (t12(ic)*t19(i+1,j,jc)+t19(i,j,jc)*t7(ic))*
          (t12(ic)*t26(i+1,j,jc)+t26(i,j,jc)*
          t7(ic))-5.0E-1*(1.5909090909090909E-6*
          (t13(i+1,j,jc)-t13(i,j,jc))+5.0E-1*(t12(ic)*
          t34(i+1,j,jc)+t34(i,j,jc)*t7(ic)))*(t12(ic)*
          t5(i+1,j,jc)+t5(i,j,jc)*t7(ic))
      122 CONTINUE
    121 CONTINUE
  120 CONTINUE
119 CONTINUE
*
ENDDO

DO 147 j=1,L
  DO 148 i=1,L
    IF (0.EQ.1-j) THEN
      HT(i,j)=1.5714285714E5*(t0(jc)*t102(i,ic,1,jc))
    ELSE
      HT(i,j)=1.5714285714E5*(t0(jc)*t102(i,ic,j,jc)+
        t102(i,ic,j-1,jc)*t2(jc))
    ENDIF
  148 CONTINUE
147 CONTINUE
*
ENDDO

Figure 2.5: Example Fortran code generated by CTADEL.
2.4 Experiments

The performance of the generated code is evaluated on an AMD 2.6 GHz Opteron with 4 GB RAM. The operating system is Scientific Linux 2.6.18. We compiled with gfortran 4.1.2 and Pathscale 3.0, both with the maximal optimization option (-O3).

The codes for the experiments are the hand-written code given in [75] and the code generated by CtADEL for the Shallow-Water equations solved by the Galerkin finite element method. The initial values were chosen the same as Steppeler [75], namely

\[
\begin{align*}
U &= \dfrac{-1}{f} \dfrac{\partial H}{\partial y}, \\
V &= \dfrac{-1}{f} \dfrac{\partial H}{\partial x}, \\
H &= H_0 + H_1 \tanh \left( \dfrac{9(y - y_0)}{2D} \right) + H_2 \dfrac{1}{\cosh^2 \left( \dfrac{9(y - y_0)}{D} \right)} \sin 2\pi x,
\end{align*}
\]

where $H_0 = 20000$, $H_1 = 4400$, $H_2 = 2660$, and $D = 4400000$. The computational domain consists of 100 × 100 grid points. After 100 time steps, the field

**Figure 2.6:** The Height field after 100 time steps
2.4. Experiments

Figure 2.7: Execution times of the hand code and the generated code compiled by the gfortran 4.1.2 (top) and the Pathscale 3.0 (down) compiler, both with -O3 optimization option.

H looks like Figure 2.6.

We verified that the CTADEL generated code reproduces the output of the hand-written code within meteorological accuracy.

To compare the performance of the CTADEL generated code with that of the hand code, we used a variety of domain sizes, and ran the model for each domain size during 100 time steps. Figure 2.7 shows the resulting execution times. We observe that both the hand code and the generated code scale very well with the number of grid points. The generated code is faster than the hand code, even strongly so for the gfortran compiled codes. This is mainly a consequence of the fact that the hand code in [73] is not well optimized, whereas CTADEL performs optimizations like common subexpression elimination. Apparently, the Pathscale compiler optimizes the hand code rather better than gfortran, but for the generated code this effect is much less pronounced.
Chapter 2. Automatic code generation for Finite Element Methods

It should be mentioned that CTADEL introduces a large number of temporary variables to store common subexpressions, and hence uses more memory, by an amount of 30%, than the hand code. This percentage can be reduced by re-using temporary variables, which feature CTADEL is currently lacking.

2.5 Conclusions

Previous studies [18, 16, 43, 42] have shown that CTADEL is able to generate efficient code for finite difference methods. We have shown that the application domain of CTADEL is not limited to finite difference methods. As an example, we applied it to the Galerkin finite element method for the Shallow-Water equations. Also for such a scheme it generates efficient (Fortran) code, that led to a faster executable than the original hand code. The speed gain depends on the compiler. It was close to a factor of 3 with an open source compiler. Furthermore, IMEX output, generated by CTADEL, provides a straightforward means to verify that the model specification matches the modeller’s intentions.

Currently, CTADEL is limited to rectangular grids, while a more complex geometry mesh is usually required in finite element methods. Though mesh generation is an important component within finite element methods, we will not include mesh generation in CTADEL, instead, we limit CTADEL to generate codes for grids produced by external mesh generators, because external mesh generators are highly optimized already for that purpose [23].